Run Inference on servers

Inference is the process of using a trained model to make predictions on new data. As this process can be compute-intensive, running on a dedicated server can be an interesting option. The `huggingface_hub` library provides an easy way to call a service that runs inference for hosted models. There are several services you can connect to:

- Inference API: a service that allows you to run accelerated inference on Hugging Face's infrastructure for free. This service is a fast way to get started, test different models, and prototype AI products.
- Inference Endpoints: a product to easily deploy models to production. Inference is run by Hugging Face in a dedicated, fully managed infrastructure on a cloud provider of your choice.

These services can be called with the InferenceClient object. It acts as a replacement for the legacy InferenceApi client, adding specific support for tasks and handling inference on both Inference API and Inference Endpoints. Learn how to migrate to the new client in the Legacy InferenceAPI client section.

> InferenceClient is a Python client making HTTP calls to our APIs. If you want to make the HTTP calls directly using your preferred tool (curl, postman,...), please refer to the Inference API or to the Inference Endpoints documentation pages.
> For web development, a JS client has been released. If you are interested in game development, you might have a look at our C# project.

Getting started

Let's get started with a text-to-image task:

```
>>> from huggingface_hub import InferenceClient
>>> client = InferenceClient()

>>> image = client.text_to_image("An astronaut riding a horse on the moon.")
>>> image.save("astronaut.png")
```

We initialized an InferenceClient with the default parameters. The only thing you need to know is the task you want to perform. By default, the client will connect to the Inference API and select a model to complete the task. In our example, we generated an image from a text prompt. The returned value is a `PIL.Image` object that can be saved to a file.

> The API is designed to be simple. Not all parameters and options are available or described for the end user. Check out this page if you are interested in learning more about all the parameters available for each task.

Using a specific model

What if you want to use a specific model? You can specify it either as a parameter or directly at an instance level:

```
>>> from huggingface_hub import InferenceClient
# Initialize client for a specific model
>>> client = InferenceClient(model="prompthero/openjourney-v4")
>>> client.text_to_image(...)
# Or use a generic client but pass your model as an argument
>>> client = InferenceClient()
>>> client.text_to_image(..., model="prompthero/openjourney-v4")
```

> There are more than 200k models on the Hugging Face Hub! Each task in the InferenceClient comes with a recommended model. Be aware that the HF recommendation can change over time without prior notice. Therefore it is best to explicitly set a model once you are decided. Also, in most cases you'll be interested in finding a model specific to *your* needs. Visit the Models page on the Hub to explore your possibilities.

Using a specific URL

The examples we saw above use the free-hosted Inference API. This proves to be very useful for prototyping and testing things quickly. Once you're ready to deploy your model to production, you'll need to use a dedicated infrastructure. That's where Inference Endpoints comes into play. It allows you to deploy any model and expose it as a private API. Once deployed, you'll get a URL that you can connect to using exactly the same code as before, changing only the `model` parameter:

```
>>> from huggingface_hub import InferenceClient
>>> client = InferenceClient(model="https://uu149rez6gw9ehej.eu-west-1.aws.endpoints.huggingface.cloud/de
# or
>>> client = InferenceClient()
>>> client.text_to_image(..., model="https://uu149rez6gw9ehej.eu-west-1.aws.endpoints.huggingface.cloud/d
```

Authentication

Calls made with the InferenceClient can be authenticated using a User Access Token. By default, it will use the token saved on your machine if you are logged in (check out how to login). If you are not logged in, you can pass your token as an instance parameter:

```
>>> from huggingface_hub import InferenceClient
>>> client = InferenceClient(token="hf_***")
```

> Authentication is NOT mandatory when using the Inference API. However, authenticated users get a higher free-tier to play with the service. Token is also mandatory if you want to run inference on your private models or on private endpoints.

Supported tasks

InferenceClient's goal is to provide the easiest interface to run inference on Hugging Face models. It has a simple API that supports the most common tasks. Here is a list of the currently supported tasks:

Domain	Task	Supported	Documentation
Audio	Audio Classification	✅	audio_classification()
	Automatic Speech Recognition	✅	automatic_speech_recognition()

Domain	Task	Supported	Documentation
	Text-to-Speech	✅	text_to_speech()
Computer Vision	Image Classification	✅	image_classification()
	Image Segmentation	✅	image_segmentation()
	Image-to-Image	✅	image_to_image()
	Image-to-Text	✅	image_to_text()
	Object Detection		
	Text-to-Image	✅	text_to_image()
Multimodal	Documentation Question Answering		
	Visual Question Answering		
NLP	Conversational	✅	conversational()
	Feature Extraction	✅	feature_extraction()
	Fill Mask		
	Question Answering		
	Sentence Similarity	✅	sentence_similarity()
	Summarization	✅	summarization()
	Table Question Answering		
	Text Classification		
	Text Generation		
	Token Classification		
	Translation		
	Zero Shot Classification		
Tabular	Tabular Classification		
	Tabular Regression		

This table is meant to be completed to support all tasks. If you are particularly interested in a task not yet supported, please let us know on huggingface_hub's repo.

> Check out the Tasks page to learn more about each task, how to use them, and the most popular models for each task.

Custom requests

However, it is not always possible to cover all use cases. For custom requests, the InferenceClient.post() method gives you the flexibility to send any request to the Inference API. For example, you can specify how to parse the inputs and outputs. In the example below, the generated image is returned as raw bytes instead of parsing it as a `PIL Image`. This can be helpful if you don't have `Pillow` installed in your setup and just care about the binary content of the image. InferenceClient.post() is also useful to handle tasks that are not yet officially supported.

```
>>> from huggingface_hub import InferenceClient
>>> client = InferenceClient()
>>> response = client.post(json={"inputs": "An astronaut riding a horse on the moon."}, model="stabilitya
>>> response.content # raw bytes
b'...'
```

Advanced tips

In the above section, we saw the main aspects of InferenceClient. Let's dive into some more advanced tips.

- The inference process takes a long time to complete.
- The model is not available, for example when Inference API is loading it for the first time.

InferenceClient has a global `timeout` parameter to handle those two aspects. By default, it is set to `None`, meaning that the client will wait indefinitely for the inference to complete. If you want more control in your workflow, you can set it to a specific value in seconds. If the timeout delay expires, an InferenceTimeoutError is raised. You can catch it and handle it in your code:

```
>>> from huggingface_hub import InferenceClient, InferenceTimeoutError
>>> client = InferenceClient(timeout=30)
>>> try:
...     client.text_to_image(...)
... except InferenceTimeoutError:
...     print("Inference timed out after 30s.")
```

Binary inputs

Some tasks require binary inputs, for example, when dealing with images or audio files. In this case, InferenceClient tries to be as permissive as possible and accept different types:

- raw `bytes`
- a file-like object, opened as binary (`with open("audio.wav", "rb") as f: ...`)
- a path (`str` or `Path`) pointing to a local file
- a URL (`str`) pointing to a remote file (e.g. `https://...`). In this case, the file will be downloaded locally before sending it to the Inference API.

```
>>> from huggingface_hub import InferenceClient
>>> client = InferenceClient()
>>> client.image_classification("https://upload.wikimedia.org/wikipedia/commons/thumb/4/43/Cute_dog.jpg/3
[{'score': 0.9779096841812134, 'label': 'Blenheim spaniel'}, ...]
```

Legacy InferenceAPI client

InferenceClient acts as a replacement for the legacy InferenceApi client. It adds specific support for tasks and handles inference on both Inference API and Inference Endpoints.

Here is a short guide to help you migrate from InferenceApi to InferenceClient.

Initialization

Change from

```
>>> from huggingface_hub import InferenceApi
>>> inference = InferenceApi(repo_id="bert-base-uncased", token=API_TOKEN)
```

to

```
>>> from huggingface_hub import InferenceClient
>>> inference = InferenceClient(model="bert-base-uncased", token=API_TOKEN)
```

Run on a specific task

Change from

```
>>> from huggingface_hub import InferenceApi
>>> inference = InferenceApi(repo_id="paraphrase-xlm-r-multilingual-v1", task="feature-extraction")
>>> inference(...)
```

to

```
>>> from huggingface_hub import InferenceClient
>>> inference = InferenceClient()
>>> inference.feature_extraction(..., model="paraphrase-xlm-r-multilingual-v1")
```

> This is the recommended way to adapt your code to InferenceClient. It lets you benefit from the task-specific methods like `feature_extraction`.

Run custom request

Change from

```
>>> from huggingface_hub import InferenceApi
>>> inference = InferenceApi(repo_id="bert-base-uncased")
>>> inference(inputs="The goal of life is [MASK].")
[{'sequence': 'the goal of life is life.', 'score': 0.10933292657136917, 'token': 2166, 'token_str': 'lif
```

to

```
>>> from huggingface_hub import InferenceClient
>>> client = InferenceClient()
>>> response = client.post(json={"inputs": "The goal of life is [MASK]."}, model="bert-base-uncased")
>>> response.json()
[{'sequence': 'the goal of life is life.', 'score': 0.10933292657136917, 'token': 2166, 'token_str': 'lif
```

Run with parameters

Change from

```
>>> from huggingface_hub import InferenceApi
>>> inference = InferenceApi(repo_id="typeform/distilbert-base-uncased-mnli")
>>> inputs = "Hi, I recently bought a device from your company but it is not working as advertised and I
>>> params = {"candidate_labels":["refund", "legal", "faq"]}
>>> inference(inputs, params)
{'sequence': 'Hi, I recently bought a device from your company but it is not working as advertised and I
```

to

```
>>> from huggingface_hub import InferenceClient
>>> client = InferenceClient()
>>> inputs = "Hi, I recently bought a device from your company but it is not working as advertised and I
>>> params = {"candidate_labels":["refund", "legal", "faq"]}
>>> response = client.post(json={"inputs": inputs, "parameters": params}, model="typeform/distilbert-base
>>> response.json()
{'sequence': 'Hi, I recently bought a device from your company but it is not working as advertised and I
```

← Search

Community Tab →